Slide 1

Administrivia

- Reminder: Homework 6 due today. Homeworks 7 and 8 posted; due next week.
- I had planned today to continue discussion of material from Chapter 4. But it would work best if most people had viewed the video lecture(s) for April 3, and (as I write this) most had not. So, some hints about Homework 7, then a digression.

  Please do watch the videos before Wednesday's class!

Slide 2

Homework 7 Help — Tracing Operation of the Processor Circuit

- In this homework, you'll trace through what the circuit in Figure 4.17 is actually doing. Examples in video lecture(s) for April 3. Idea is for you to trace through what the circuit actually does rather than what you think it should do. But the two should match!
- (To be continued . . .)

Slide 3

Memory Hierarchy — Overview

- Significant overlap between Chapter 5 and material covered in operating-systems course (as I teach it anyway). In previous years pretty much all students went on to that course. Now not all do. Either way, not a bad idea to discuss briefly now.
- A key idea (borrowed from one writer of O/S textbooks): In a perfect world, we could have as much memory as we wanted, and it would be very fast and very cheap. In the real world, there are tradeoffs (e.g., fast versus cheap, fast versus large).

Slide 4

“Principle of Locality”

- Basic underlying idea: Most applications exhibit locality with regard to memory.
- “Temporal locality”: Memory locations referenced in the near past likely to be referenced again in the near future. (At any given time, a program isn’t going to be working with *all* of its data.)
- “Spatial locality”: Memory locations close together in space likely to be referenced close together in time. (Examples include processing arrays sequentially; accessing local variables, which are apt to be located together in memory).

Slide 5

Memory Hierachy and Caching

- To exploit temporal locality, can use “caching”: Keep copies of frequently-used data in faster but smaller memory. Can do this on multiple levels.
- To exploit spatial locality, can move data between levels in blocks.
- *Note* that while impact of caching on performance can be significant, it should not affect results (which is why it makes some sense to just ignore it initially).

Slide 6

Caches (Between Processor and RAM) — Executive-Level Summary

- Idea here is to interpose a “cache” (small but fast) between the processor and the memory, and use it to hold frequently-referenced data, and have this managed mostly by the hardware.
- Read “from memory” tries cache first, and then if not found there goes to RAM and updates cache.
- Write “to memory” is maybe more interesting: Writes to cache, but then must at some point write to RAM also — maybe right away (easier to get right but can be slow) or later.

Slide 7

Virtual Memory — Executive-Level Summary

- Basic idea here is to fake having more RAM than you really have, by keeping some data that would be in RAM on disk. In a sense, RAM is a cache for the "real" memory, on disk(!).
- It turns out that a common way to do this also facilitates protecting one process's data from other processes.

Slide 8

Cache Coherence — Executive-Level Summary

- Clearly(?) possible for cached data to be out of synch with data in memory. Probably of most concern if multiple processing elements, each with its own cache, share memory.
- Various schemes exist for ensuring that programs don't have to be aware of this complication. Details in the textbook.

Slide 9

Caches and Applications Programming

- Mostly, memory hierarchy (including virtual memory) managed transparently by combination of hardware and (operating-system) software.

  So first approximation presented in introductory courses (memory essentially a really big array of bytes, with addresses as indices) is okay, especially if you just want right answers.

- However, effects on performance can be significant, so if you want right answers fast . . .

Slide 10

Caches and Applications Programming, Continued

- For single-threaded programs, key idea is to maximize locality (temporal and spatial). Rearranging order in which data is accessed can have a big effect. (Matrix-multiplication example.)

- For multi-threaded programs, also need to consider whether multiple threads need to share access to the same data (problem for correctness too!) or even nearby data (“false sharing” — no effect on correctness but can be slow).

Slide 11

Minute Essay

- None — quiz.